การจัดการหน่วยความจำ ของ 80386 (การแบ่งเซกเมนต์)

ถวัลย์ ตั้งลิตานนท์

ในบทความพิเศษของฉบับที่ 48 ได้กล่าวถึงลักษณะโดยทั่วไปของ 80386 เช่นโครงสร้างภายใน การทำงานในโหมดต่างๆ ชุดคำสั่ง การจัดการหน่วยความจำ และอื่น ๆ ซึ่งในเรื่องของการจัดการหน่วยความจำยังแบ่งออกเป็น การแบ่งเซกเมนต์ การทำเพจจิ้ง และหน่วยความจำแคช

ในฉบับนี้จะขอกล่าวถึงรายละเอียดของการแบ่งเซกเมนต์ว่า 80386 มีการแบ่งเซกเมนต์อย่างไร เซกเมนต์มีกี่ชนิด และสามารถอ้างถึงได้อย่างไร ซึ่งเราจะได้ติดตามกันต่อไป

นิยามของเซกเมนต์

ในการแบ่งหน่วยความจำเป็นเซกเมนต์ที่ใช้ใน 80386 เราไม่สามารถที่จะแทนข้อมูลทุกข้อมูลที่กำหนดเซกเมนต์ในเซกเมนต์รีจิสเตอร์ขนาด 16 บิตได้ โดยเฉพาะอย่างยิ่งเมื่อมีการแบ่งเป็นระดับสิทธิพิเศษและมีการทำงานร่วมกัน การป้องกันเป็นสิ่งที่จำเป็นมากขึ้น กลไกบางอย่างนอกเหนือจากเซกเมนต์รีจิสเตอร์ขนาด 16 บิตจะต้องถูกนำมาใช้ ดังนั้น 80386 จึงใช้วิธีการกำหนดหน่วยความจำแบบเซกเมนต์ที่แตกต่างไปจากเดิมที่ใช้ใน 8086 แต่ยังคงบางส่วนที่คอมแพทติเบิ้ลกับตัวโปรเซสเซอร์ตระกูลที่ลงท้ายด้วยเลข 86 ตัวก่อนหน้า

ความแตกต่างพื้นฐานเกือบทั้งหมดในการอ้างหน่วยความจำแบบเซกเมนต์ของ 80386 ก็คือ โปรแกรมของเราไม่สามารถจะกระทำกับหน่วยความจำใด ๆ เพียงแค่การอ้างถึงตำแหน่งนั้น สำหรับผู้ที่เคยใช้งาน 8086 จะทราบดีว่าตัวโปรเซสเซอร์สามารถอ้างแอดเดรสได้ 1 เมกะไบต์ ซึ่งแบ่งออกเป็นเซกเมนต์ละ 64 กิโลไบต์เรียงกันไปตามลำดับ เราสามารถอ้างถึงหน่วยความจำตำแหน่งใด ๆ ก็ตามโดยการโหลดเซกเมนต์รีจิสเตอร์ด้วยค่าเบสแอดเดรสและชี้เข้าไปในบริเวณเซกเมนต์นั้นด้วยค่าออฟเซตจาก 0 ถึง FFFF เพื่อที่จะกระทำกับตำแหน่งในส่วนบนหรือส่วนล่างก็ตามของหน่วยความจำ ถ้าต้องการเข้าถึงหน่วยความจำทั้ง 1 เมกะไบต์ เราจะต้องเลื่อนเซกเมนต์รีจิสเตอร์ไปอย่างน้อย 16 ครั้ง (16 × 64 กิโลไบต์ = 1 เมกะไบต์)

เป็นที่น่าสังเกตอย่างหนึ่งว่าการใช้เซกเมนต์ในแบบของ 8086 นั้น เราไม่มีวิธีการใด ๆ ในการป้องกันหน่วยความจำเลย ยกเว้นแต่ใช้ฮาร์ดแวร์ภายนอกเท่านั้น ยิ่งไปกว่านั้นก็คือหน่วยความจำฟิซิคัลที่มีอยู่ก็ไม่สัมพันธ์กับสถาปัตยกรรมชนิดนี้ ซึ่งก็คาดว่านักเขียนโปรแกรมจะไม่อ้างแอดเดรสซึ่งไม่สอดคล้องกับหน่วยความจำฟิซิคัล ถ้าโปรแกรมมีการอ้างถึงตำแหน่งของหน่วยความจำที่ไม่มีอยู่จริง ตัวโปรเซสเซอร์ก็จะอ่านข้อมูลที่ไม่ถูกต้องเข้ามาทำให้ระบบไม่ทำงาน ทั้งนี้เป็นเพราะขึ้นอยู่กับการออกแบบการถอดรหัสของฮาร์ดแวร์นั่นเอง

การโปรแกรมในระบบของ 80386 นี้ (ไม่ว่าจะอยู่ในระดับใดก็ตาม) จะไม่สามารถกระทำกับพื้นที่ของหน่วยความจำนั้นได้จนกว่าพื้นที่นั้นจะถูกบอกให้โปรแกรมรับรู้ โปรแกรมไม่สามารถจะอ้างถึงทุก ๆ แอดเดรสได้ และโปรแกรมที่พยายามจะผ่านไปยังหน่วยความจำทั้ง 4 กิกะไบต์โดยการเพิ่มค่าอินเด็กซ์ไปเรื่อย ๆ จะเกิดเอกเซพชัน (EXCEPTION) ขึ้น ซึ่งเกิดจากการละเมิดออกนอกเซกเมนต์ รูปจำลองของแอดเดรสลักษณะนี้จะเป็นพื้นฐานของสถาปัตยกรรมของ 80386 ที่ถูกสร้างขึ้น ซึ่งครอบคลุมไปถึงการแบ่งเซกเมนต์ การป้องกันและคุณลักษณะของมัลติทาสกิ้ง

อย่างที่กล่าวมา เราจะเห็นแนวความคิดของเซกเมนต์ของ 80386 ที่ใช้งานในโปรเทคโหมดจะต่างกันมากกับ 8086 ในสถาปัตยกรรมของ 80386 นิยามของคำว่าเซกเมนต์จึงยืดหยุ่นมาก ตัวโปรเซสเซอร์ไม่สามารถจะจำกัดขอบเขตของเซกเมนต์อย่างเช่นใน 8086 ได้ แต่ 80386 มีกลไกให้นักเขียนโปรแกรมระบบสามารถกำหนดว่าแต่ละเซกเมนต์

ควรอยู่ที่ใด รวมทั้งแอดเดรสเริ่มต้น ความยาว จุดมุ่งหมายในการใช้ และคุณสมบัติประจำตัวอื่น ๆ ของเซกเมนต์

การกำหนดเซกเมนต์สามารถทำได้เกือบทุกอย่างที่เราต้องการ ตัวอย่างเช่น เซกเมนต์ไม่จำเป็นต้องยาว 64 กิโลไบต์ เราสามารถกำหนดให้มีความยาวเท่าไรก็ได้จาก 1 ไบต์ถึง 4 กิกะไบต์ เซกเมนต์ขนาด 1 ไบต์ก็คือเซกเมนต์ที่มีค่าออฟเซตที่ถูกต้องเพียงค่าเดียว ตรงกันข้ามเซกเมนต์ขนาด 4 กิกะไบต์ซึ่งจะครอบคลุมบริเวณของแอดเดรสของ 80386 ทั้งหมดอย่างเชิงเส้นจาก 0 ถึง FFFF-FFFF ซึ่งหมายความว่าเราสามารถใช้รีจิสเตอร์ขนาด 32 บิตเป็นตัวชี้แอดเดรสได้ตลอดบริเวณโดยไม่ต้องเปลี่ยนค่าเซกเมนต์รีจิสเตอร์

ค่าของเซกเมนต์ที่ถูกกำหนดให้กับแอดเดรสเริ่มต้น (เบสแอดเดรส) สามารถจะเป็นแอดเดรสใด ๆ ก็ได้ ในบริเวณที่เป็นเชิงเส้นขนาด 32 บิต ถ้าเปรียบเทียบกับ 8086 ก็คือค่าที่จะโหลดให้กับเซกเมนต์รีจิสเตอร์นั่นเอง เบสแอดเดรสสามารถถูกกำหนดเป็นแอดเดรสจริงที่เป็นเชิงเส้นซึ่งจะถูกกระทำเมื่อโปรแกรมอ้างถึงออฟเซตที่เป็น 0 ในเซกเมนต์ ทำให้ไม่จำเป็นต้องมีการปรับค่า 16 ไบต์อย่างเช่น 8086 การกำหนดเซกเมนต์วิธีนี้จึงทำให้นักเขียนโปรแกรมสามารถควบคุมการใช้หน่วยความจำได้อย่างสมบูรณ์

เซกเมนต์จะถูกกำหนดให้มีคุณสมบัติประจำตัว คุณสมบัติอันหนึ่งกำหนดว่าเซกเมนต์จะถูกใช้สำหรับอะไร เช่น โค้ด ข้อมูล สแต็กหรือจุดประสงค์อื่น ส่วนคุณสมบัติอีกอันหนึ่ง เป็นตัวกำหนดระดับชั้นพิเศษ (privilege level) จาก 0 ถึง 3 ซึ่งในเรื่องของระดับชั้นพิเศษจะได้นำเสนอในโอกาสต่อไป

สรุปโดยย่อเซกเมนต์ก็คือ พื้นที่ของหน่วยความจำซึ่งถูกกำหนดจากนักเขียนโปรแกรมใช้สำหรับจุดมุ่งหมายที่แตกต่างกัน เช่น โค้ด ข้อมูล หรือสแต็ก เซกเมนต์ไม่จำเป็นต้องมีขนาดเท่ากันหรือถูกจำกัดขนาดอยู่ที่ 64 กิโลไบต์

เซกเมนต์เดสคริบเตอร์

เซกเมนต์หนึ่ง ๆ จะถูกบรรยายลักษณะของเซกเมนต์ด้วยเซกเมนต์เดสคริบเตอร์ ในระบบหนึ่ง ๆ ที่กำลังรันอยู่ แต่ละเซกเมนต์ของหน่วยความจำจะมีเซกเมนต์เดสคริบเตอร์เพียงหนึ่งเดียวเท่านั้น เซกเมนต์เดสคริบเตอร์จะบ่งบอกถึงเบสแอดเดรส ความยาว ชนิด ระดับชั้นพิเศษ และข้อมูลเกี่ยวกับสถานภาพต่าง ๆ ของเซกเมนต์ ดังรูปที่ 1

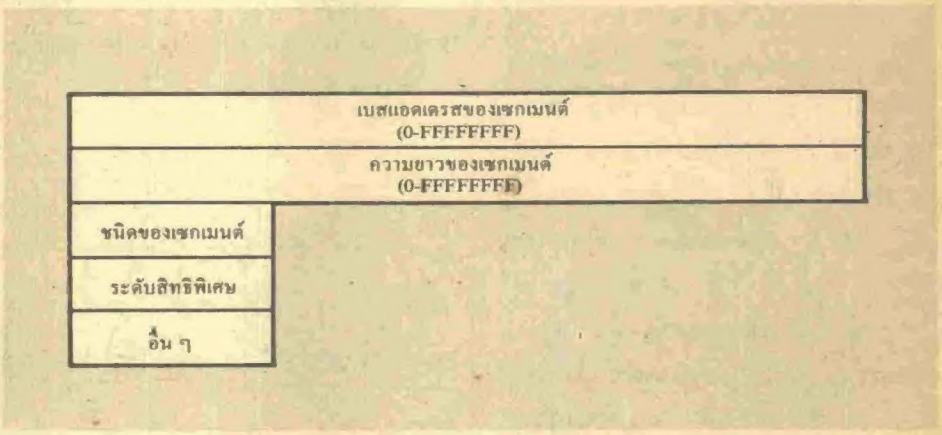
เซกเมนต์เดสคริบเตอร์จะบ่งบอกทุกอย่างที่ควรจะรับรู้เกี่ยวกับเซกเมนต์ของหน่วยความจำ เมื่อเราสร้างเซกเมนต์เดสคริบเตอร์ เราต้องกำหนดพื้นที่ของแอดเดรสที่จะใช้ภายใต้ข้อจำกัดของนิยามของเดสคริบเตอร์ ตัวอย่างเช่น เราสามารถกำหนดให้เซกเมนต์มีช่วงของแอดเดรสจาก A400 ถึง CE04 และเซกเมนต์อื่น ๆ ครอบคลุมจากแอดเดรส 230000 ถึง 230FFF ได้

พึงระลึกเสมอว่าถ้าเราไม่ได้บ่งบอกบริเวณของแอดเดรสในเดสคริบเตอร์ บริเวณแอดเดรสนั้นจะไม่สามารถอ้างแอดเดรสถึงได้ ตัวโปรเซสเซอร์จะไม่ยอมกระทำกับบริเวณนั้น ซึ่งเป็นข้อดีอย่างยิ่งในการรักษาความปลอดภัยของระบบ

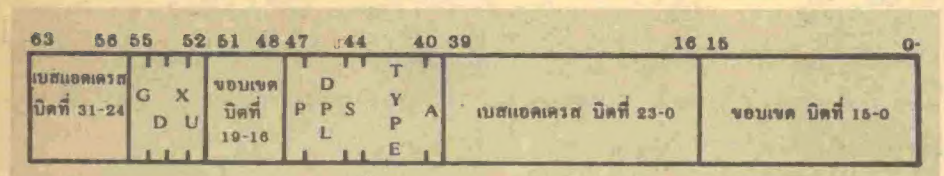
รูปแบบของเซกเมนต์เดสคริบเตอร์จะไม่อยู่ในรูปที่ง่าย ๆ ดังรูปที่ 1 รูปแบบจริง ๆ ที่เป็นต้นแบบจะเป็นดังรูปที่ 2 บิตที่ 16 ถึง 39 จะเก็บ 24 บิตล่างของเบสแอดเดรสของเซกเมนต์ ส่วน 8 บิตบนจะเก็บอยู่ที่บิต 56 ถึง 63 ตัวโปรเซสเซอร์จะนำ 2 ส่วนนี้มาต่อกันเองและอ้างเป็นจุดเริ่มต้นของเซกเมนต์

อีกส่วนหนึ่งที่มีความสำคัญมากก็คือ ลิมิตฟิลด์ ซึ่งแบ่งออกเป็น 2 ส่วน อินเทลได้กำหนดให้ลิมิตฟิลด์เป็นความยาวของเซกเมนต์ (ในหน่วยของไบต์) ลบด้วยหนึ่ง ตัวอย่างเช่น เซกเมนต์ที่ยาว 1 ไบต์ก็จะมีค่าในลิมิตเป็น 0 เซกเมนต์ที่มีขนาด 4 กิกะไบต์ก็จะมีลิมิตเท่ากับ FFFFFFFF เพื่อที่จะให้จำได้ง่าย ก็คือ ค่าในลิมิตฟิลด์เป็นเท่าไร ค่าออฟเซตที่ใช้อ้างแอดเดรสก็เป็นเท่านั้น (แต่อย่าลืมด้วยค่าออฟเซตเริ่มต้นที่ 0)

บิต 0 ถึง 15 ของเดสคริบเตอร์จะเก็บค่า 16 บิตล่างของลิมิต ส่วน 4 บิตบนจะเก็บที่บิต 48 ถึง 51 จะเห็นว่าค่าลิมิตมีจำนวน 20 บิต ซึ่งก็คือ 1 เมกะไบต์ (2 ยกกำลัง 20) เท่านั้น แต่จากที่



รูปที่ 1 ลักษณะของเซกเมนต์เดสคริบเตอร์



ชนิดของเซกเมนต์			
000	ข้อมูล	อ่านได้เท่านั้น	
001	ข้อมูล	อ่านได้/เขียนได้	
010	สแต็ก	อ่านได้เท่านั้น	
011	สแต็ก	อ่านได้/เขียนได้	
100	โค้ด	เอ็กซีคิวต์ได้เท่านั้น	
101	โค้ด	เอ็กซีคิวต์ได้/อ่านได้	
110	โค้ด	เอ็กซีคิวต์ได้เท่านั้น	conforming
111	โค้ด	เอ็กซีคิวต์ได้/อ่านได้	conforming

รูปที่ 2 รูปแบบของเซกเมนต์เดสคริปเตอร์

เราทราบมาข้างต้นขนาดของเซกเมนต์ได้ถูกจำกัดอยู่เพียง 1 เมกะไบต์ เพราะฉะนั้นบิตที่เหลือในเซกเมนต์เดสคริปเตอร์จะเป็นตัวอธิบายให้เรากระจ่างในเรื่องนี้

เรามาทำความเข้าใจกับบิตที่ 40 ถึง 55 ในเซกเมนต์เดสคริปเตอร์ดังนี้

A(Accessed, บิตที่ 40) : 80386 จะทำการเซตบิตนี้โดยอัตโนมัติเมื่อใดก็ตามที่หน่วยความจำซึ่งถูกกำหนดในเดสคริปเตอร์นี้ถูกอ้างถึง

TYPE (Segment type, บิตที่ 41 ถึง 43) : 3 บิตนี้แสดงชนิดของเซกเมนต์ซึ่งเรากำหนดให้ โดยทั่วไปชนิดของเซกเมนต์ก็คือ โค้ด ข้อมูล และสแต็ก

S(System, บิตที่ 44) : ถ้าบิตนี้ถูกเคลียร์จะหมายถึง เซกเมนต์เดสคริปเตอร์นี้เป็นของระบบ ถ้าบิตนี้ถูกเซตจะหมายถึงเซกเมนต์เดสคริปเตอร์ไม่ใช่ของระบบ (โค้ด ข้อมูล หรือสแต็ก)

DPL (Descriptor Privilege level, บิตที่ 45 และ 46) : 2 บิตนี้จะแสดงระดับสิทธิพิเศษของพื้นที่หน่วยความจำ ซึ่งกำหนดในเดสคริปเตอร์ DPL = 0 หมายถึงสิทธิพิเศษสูงสุด DPL = 3 คือต่ำสุด

P (Present, บิตที่ 47) : ถ้าบิตนี้ถูกเคลียร์ หมายถึงช่วงของแอดเดรสที่กำหนดในเดสคริปเตอร์จะถูกพิจารณาเป็นหน่วยความจำชั่วคราว ไม่ใช่หน่วยจำฟิซิคัล

U(User, บิตที่ 52) : บิตนี้ไม่ถูกกำหนดอย่างสมบูรณ์ และ 80386 จะไม่สนใจบิตนี้ ผู้ใช้สามารถใช้บิตนี้ได้อย่างอิสระ

X(Reserved by Intel, บิตที่ 53) : บิตนี้ตรงข้ามกับบิตที่ 52 บิตนี้จะต้องไม่ถูกกำหนด ซอฟต์แวร์ของเราต้องไม่ใช้บิตนี้

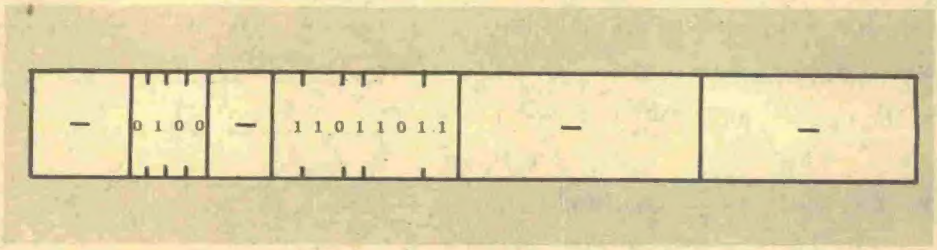
D(Default size, บิตที่ 54) : เมื่อบิตนี้ถูกเคลียร์ โอเปอแรนด์ที่อยู่ในเซกเมนต์นี้จะถูกสมมติว่ามีขนาด 16 บิต ถ้าบิตนี้ถูกเซตโอเปอแรนด์ก็จะถูกสมมติว่ามีขนาด 32 บิต

G(Granularity, บิตที่ 55) : เมื่อบิตนี้ถูกเคลียร์ ลิมิตฟิลด์ 20 บิตจะถูกสมมติว่าถูกวัดในหน่วยของไบต์ ถ้าบิตนี้ถูกเซต ลิมิตฟิลด์จะถูกสมมติว่าถูกวัดในหน่วยของ 4096 ไบต์ (4 กิโลไบต์)

ตัวอย่างของเซกเมนต์เดสคริปเตอร์ที่ถูกต้องแสดงดังรูปที่ 3 บิตที่ 40 ถึง 47 หรือไบต์ที่ 6 ของเดสคริปเตอร์ครอบคลุม 5 ฟิลด์แรกที่ได้กล่าวไปข้างต้น ค่าในไบต์นี้ไม่สามารถเป็นไปได้ทั้ง 256 ค่า ตัวอย่างเช่น ค่า 00H และ 80H อินเทลสงวนไว้ ถ้าเราพยายามจะให้เดสคริปเตอร์ที่มีค่าของไบต์นี้ไม่ถูกต้อง 80386 จะฟ้องว่ามี fault เกิดขึ้น (ดูรูปที่ 3)

บิต Accessed ของเดสคริปเตอร์จะถูกเซตโดยอัตโนมัติเมื่อใดก็ตามที่เซกเมนต์ของหน่วยความจำที่กำหนดในเดสคริปเตอร์นั้นถูกใช้ ตัวโปรเซสเซอร์จะไม่เคลียร์บิตนี้เลย ซึ่งทำให้เราสามารถหาข้อมูลบางอย่างได้ เช่น ความถี่ในการใช้หน่วยความจำนั้น โดยเราสามารถเขียนโปรแกรมที่คอยทดสอบและเคลียร์บิตนี้อย่างสม่ำเสมอในแต่ละเดสคริปเตอร์ จำนวนครั้งที่บิตนี้ถูกเซตก็คือข้อมูลที่บอกว่าหน่วยความจำนั้นถูกใช้บ่อยเท่าไรนั่นเอง

บิต Present เป็นตัวบอกโปรเซสเซอร์ว่าช่วงของแอดเดรสที่อยู่ในเดสคริปเตอร์จะถูกพิจารณาว่าเป็นหน่วยความจำฟิซิคัล บิตนี้มีความสำคัญมาก คือ เปรียบเสมือนแฟล็กที่บอกว่าเดส-



รูปที่ 3 ตัวอย่างของเซกเมนต์เดสคริปเตอร์

ดริบเตอร์นี้ถูกต้อง ขณะที่เราสร้างเดสคริบเตอร์ ถ้าเราไม่เซตบิต Present นี้ 80386 จะไม่ยอมให้ใช้คำสั่งกับเดสคริบเตอร์นี้ บริเวณของแอดเดรสที่กำหนดในเดสคริบเตอร์จะไม่สามารถถูกใช้ได้เลย และพึงจำไว้ว่าถ้าเราไม่เซตบิต Present นี้ โปรเซสเซอร์จะไม่รับรู้ส่วนที่เหลืออื่น ๆ ในเดสคริบเตอร์เลย ดังรูปที่ 4

บิต Default เป็นคุณลักษณะอันหนึ่งซึ่งใช้ทั่วไปในสถาปัตยกรรมของ 80386 ซึ่งทำให้คอมแพตกับโปรเซสเซอร์ตัวก่อนหน้านี้ คุณลักษณะนี้ยอมให้เรารันโค้ดของ 80286 ในโปรเทคโหมดของ 80386 ถ้าเซกเมนต์ถูกกำหนดให้เก็บโค้ด และบิต Default ถูกเคลียร์ โปรเซสเซอร์จะแปลสิ่งที่อยู่ในเซกเมนต์ว่าเป็นโค้ด 16 บิตของ 80286 แต่ถ้าบิต Default ถูกเซตค่าภายในเซกเมนต์จะมองเป็นโค้ด 32 บิตของ 80386 สำหรับเซกเมนต์ของข้อมูลและสแต็กก็เช่นเดียวกัน เมื่อ D = 0 การกระทำกับสแต็กจะกระทำทีละ 16 บิต SP จะถูกใช้เป็นสแต็กพอยน์เตอร์แทน ESP และขนาดของสแต็กจะมีค่ามากที่สุดเท่ากับ FFFF (64 กิโลไบต์) แทนที่จะเป็น FFFFFFFF (4 กิกะไบต์)

บิต Granularity ทำให้เราสามารถสร้างขนาดของเซกเมนต์ที่ใหญ่กว่า 1 เมกะไบต์ได้ เนื่องจากแต่ละเดสคริบเตอร์ มีที่ว่างไม่เพียงพอสำหรับเก็บลิมิตฟิลด์ทั้ง 32 บิตได้ ดังนั้นจึงใช้เพียง 20 บิต ซึ่งเพียงพอสำหรับลิมิตขนาด 1 เมกะไบต์โดยมีความละเอียด 1 ไบต์ จากจุดนี้จะเห็นว่าลิมิตฟิลด์สามารถถูกแปลงเป็นหน่วยของ 4 กิโลไบต์ได้ ถ้าบิต G ถูกเซต (G = 1) 80386 จะทำการเลื่อนลิมิตฟิลด์ไปทางซ้าย 12 บิต และเติม 1 เข้าไปแทน ดังนั้นถ้า G = 1 ลิมิตฟิลด์ที่มีค่า 400 จะกลายเป็น 400FFF ถึงแม้ว่าจะไม่ได้ความละเอียดถึง 1 ไบต์ แต่ถ้าเรากำหนดขนาดของเซกเมนต์ที่ใหญ่กว่า 1 เมกะไบต์ ก็จะไม่มีปัญหาแต่อย่างไรกับขนาดเพียงไม่กี่กิโลไบต์ที่เกินมา

มาถึงจุดนี้จะขอสรุปก่อนว่าเซกเมนต์เดสคริบเตอร์เป็นตัวกำหนดสิ่งต่าง ๆ ต่อไปนี้

– เบสแอดเดรส (32 บิต)

– ค่าลิมิต (ความยาว) ของเซกเมนต์ (20 บิต)

– ชนิดของเซกเมนต์ (4 บิต)

– ระดับสิทธิพิเศษ (2 บิต)

– กำหนดว่าเมื่อไรที่เซกเมนต์จะถูกพิจารณาว่าเป็นฟิซิคัล (1 บิต)

– กำหนดว่าเซกเมนต์เคยถูกแอกเซสมาก่อนหรือไม่ (1 บิต)

– ขนาดของเซกเมนต์ ให้มีขนาดสูงสุด 1 เมกะไบต์ หรือ 4 กิกะไบต์ (1 บิต)

– ขนาดของโอเปอแรนด์ที่อยู่ในเซกเมนต์ (1 บิต)

– แฟล็กที่อินเทลสงวนไว้ (1 บิต)

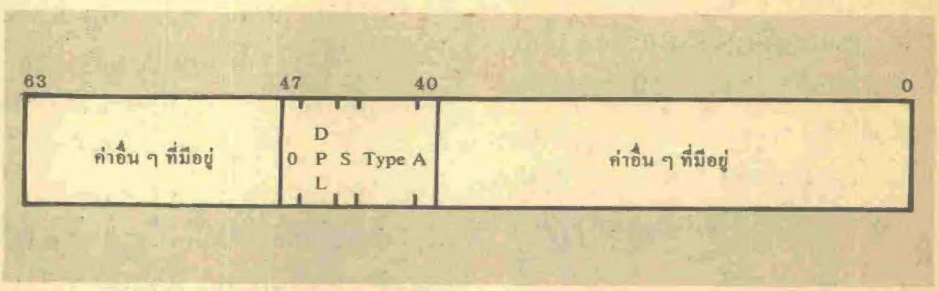
– แฟล็กสำหรับผู้ใช้ (1 บิต)

จากรูปที่ 2 ซึ่งแสดงชนิดของเซกเมนต์โดยใช้ 3 บิต คือ บิตที่ 41 ถึง 43 ทำให้เราสามารถกำหนดชนิดของเซกเมนต์ได้ 8 รูปแบบที่ต่างกัน เราจะมาดูว่ารูปแบบทั้ง 8 นี้มีรายละเอียดอะไรบ้าง

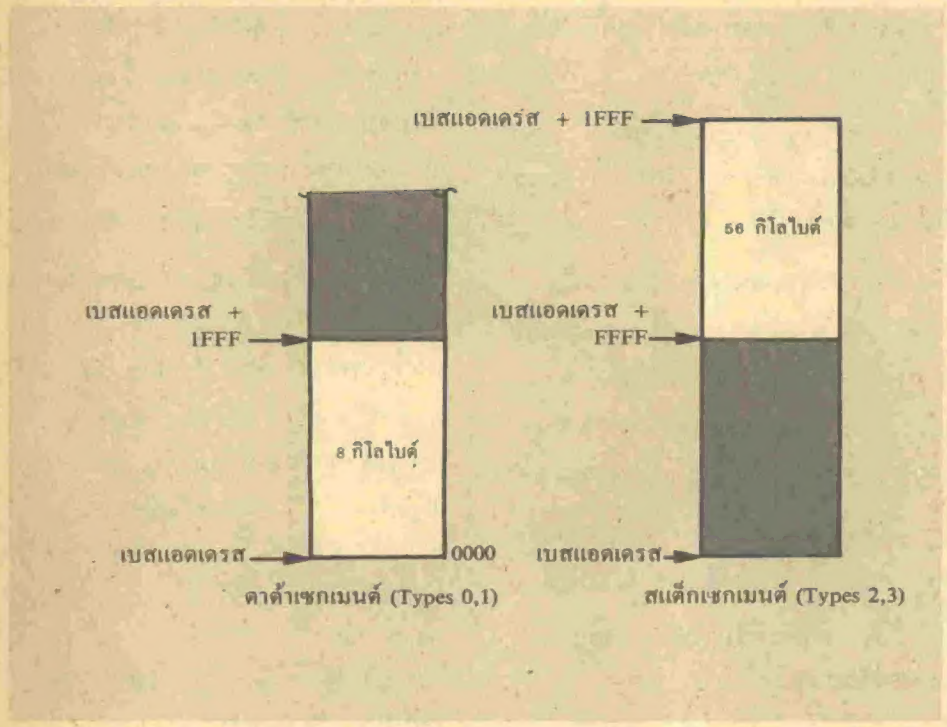
Type = 0 (000) จะเป็นตัวกำหนดว่าเซกเมนต์นี้ยอมให้อ่านข้อมูลเท่านั้นไม่ยอมให้เขียน ซึ่งเสมือนกันว่ากำหนดเป็นพื้นที่ของรอม 80386 จะไม่ยอมทำคำสั่งใด ๆ เกี่ยวกับการเขียนหน่วยความจำเมื่ออ้างถึงเซกเมนต์นี้

Type = 1 (001) ก็คล้ายกับ Type = 0 ต่างกันตรงที่ Type นี้ยอมให้เขียนลงหน่วยความจำได้ และโดยทั่วไปมักจะใช้เป็นดาต้าเซกเมนต์ การอ่าน/เขียนก็เพื่อใช้เก็บข้อมูลเท่านั้น เราไม่สามารถเก็บโค้ดไว้ในเซกเมนต์นี้ได้ เพราะว่า 80386 จะไม่ยอมเฟตช์โค้ดจากเซกเมนต์ซึ่งไม่ถูกกำหนดก่อนว่ายอมให้เอ็กซีคิวท์ได้

Type = 2 หรือ 3 กำหนดสแต็กเซกเมนต์ ซึ่งจะแตกต่างจากดาต้าเซกเมนต์ตรงที่ว่าลิมิตฟิลด์ของสแต็กเซกเมนต์จะเป็นตัวกำหนดบริเวณที่ไม่สามารถอ้างแอดเดรสได้ รูปที่ 5 จะแสดงดาต้าเซกเมนต์และสแต็กเซกเมนต์ซึ่งมีค่าลิมิตเท่ากับ 1FFF ในกรณีของดาต้าเซกเมนต์ ออฟเซตที่ถูกต้องจะต้องมีค่าน้อยกว่าหรือเท่ากับค่าของลิมิต แอดเดรสแรกที่สามารถอ้างได้ก็คือ แอดเดรสที่มีออฟเซตเป็น 0 และแอดเดรสสุดท้ายคือออฟเซตเป็น 1FFF ดาต้าเซกเมนต์จึงมีขนาด 2000 ไบต์ (8 กิโลไบต์) ในกรณีของสแต็กเซกเมนต์ ค่าออฟเซตที่ถูกต้องคือค่าที่มากกว่าค่าลิมิต แอดเดรสแรกคือแอดเดรสที่มีออฟเซต



รูปที่ 4 แสดงเซกเมนต์เดสคริบเตอร์ที่ไม่สามารถใช้งานได้



รูปที่ 5 ดาต้าเซกเมนต์และสแต็กเซกเมนต์

เป็น 2000 และสุดท้ายคือ FFFF ทำให้สแต็กเซกเมนต์มีขนาด E000 ไบต์ (56 กิโลไบต์) ดังรูปที่ 5

สำหรับสแต็กเซกเมนต์ ขอบเขตล่างของเซกเมนต์คือค่าในลิมิตฟิลด์ ส่วนขอบเขตบนนั้นเป็นไปได้ 2 ค่า ถ้า D (บิต 54) = 0 ขอบเขตบนจะเท่ากับเบสแอดเดรสบวกกับค่าคงที่ FFFF (64 กิโลไบต์) แต่ถ้า D = 1 ขอบเขตบนจะเท่ากับเบสแอดเดรสบวกกับค่าคงที่ FFFF FFFF (4 กิกะไบต์) เนื่องจากค่าออฟเซตจะต้องมากกว่าค่าในลิมิตฟิลด์ ดังนั้นสแต็กเซกเมนต์จะมีค่ามากที่สุด เมื่อค่าในลิมิตฟิลด์เป็น 0 และมีค่าน้อยที่สุดเมื่อค่าในลิมิตฟิลด์เป็น FFFF (D = 0) หรือ FFFFFFFF (D = 1) นั่นคือขนาดของสแต็กเซกเมนต์มีความยาวเป็น 0

อีกเรื่องหนึ่งที่พึงระลึกถึงก็คือเมื่อเราเซตบิต D ซึ่งหมายความว่าสแต็กเซกเมนต์จะถูกกระทำทีละ 32 บิต (ใช้รีจิสเตอร์ ESP) และทำให้ขอบเขตบนมีค่าเป็น 4 กิกะไบต์

ส่วน Type 4 5 6 และ 7 กำหนดว่าเป็นเซกเมนต์ที่สามารถเอ็กซิคิวท์ได้ นั่นก็คือเซกเมนต์นี้จะเป็นที่เก็บของโค้ด สังเกตว่าการยอมให้มีการอ่านข้อมูลได้ (Type 5) ทำให้เราสามารถเก็บค่าคงที่ร่วมกับโค้ด ส่วน Type 4 จะไม่ยอมให้มีการอ่านข้อมูล สามารถเฟตช์โค้ด (คำสั่ง) ได้อย่างเดียว ซึ่งเป็นประโยชน์อย่างมากในการป้องกันการดูออปเจ็คโค้ดจากบุคคลอื่น ส่วนอีก 2 ชนิดคือ Type 6 และ 7 มีคุณสมบัติประจำตัวที่เรียกว่า comforming ซึ่งมีความหมายง่าย ๆ ก็คือโค้ดเซกเมนต์นี้จะยอมละทิ้งการป้องกันที่เป็นแบบระดับสิทธิพิเศษ ซึ่งจะไม่ขอกล่าวรายละเอียดในที่นี้

ตารางเดสคริบเตอร์

อย่างที่ทราบมาแล้วข้างต้น เราสามารถจะสร้างเซกเมนต์เดสคริบเตอร์มากเท่าไรก็ได้ตามที่เราต้องการ และสามารถเพิ่มเดสคริบเตอร์ในฐานะที่เป็นของระบบ หรือเป็นของผู้ใช้ ตอนนี้เราจะดูกันว่าตัวโปรเซสเซอร์จัดการอย่างไรกับการแยกเป็นส่วน ๆ ซึ่งเป็นส่วนของข้อมูลที่ไม่มีความสัมพันธ์กัน

เซกเมนต์เดสคริบเตอร์ที่เรากำหนดขึ้นจะต้องถูกจัดเป็นกลุ่มและถูกวางต่อ ๆ กันในหน่วยความจำ การจัดเป็นกลุ่มนี้เรียกว่าตารางเดสคริบเตอร์ ลำดับของการจัดวางเดสคริบเตอร์จะไม่มีความสำคัญ ตารางนี้จะมีเพียงหนึ่งเดสคริบเตอร์หรือมีมากถึง 8192 ซึ่งทำให้ตารางมีความยาว 64 กิโลไบต์ก็ได้ ตารางเดสคริบเตอร์จะมี 3 ชนิด แต่ละชนิดก็จะถูกใช้โดยจุดประสงค์ที่แตกต่างกัน ซึ่งเราจะต้องพิจารณาจุดมุ่งหมายในการใช้ก่อนที่จะตัดสินใจ

ต่อไปจะได้กล่าวถึงชนิดของตารางเดสคริบเตอร์ ซึ่งอาจจะต้องใช้ศัพท์ภาษาอังกฤษบ้างเพื่อความสะดวกในการอ้างถึง ตารางหลักของเดสคริบเตอร์ที่เป็นตารางเอนกประสงค์ก็คือ global descriptor table (GDT) GDT สามารถถูกใช้โดยโปรแกรมทุกโปรแกรมเพื่ออ้างเซกเมนต์ของหน่วยความจำ interrupt descriptor table (IDT) จะเก็บเซกเมนต์เดสคริบเตอร์ที่กำหนดอินเตอร์รัพต์หรือเอ็กเซพชันรูทีน IDT สามารถแทนตารางของอินเตอร์รัพต์เวกเตอร์ที่ใช้ใน 8086 ได้โดยตรง 80386 ที่ทำงานในโปรเทคโหมดจะต้องกำหนด GDT และ IDT อย่างละ 1 ตารางอย่างแน่นอน

ตารางชนิดสุดท้ายคือ local descriptor table (LDT) สามารถถูกกำหนดการใช้ต่อหนึ่งงานในระบบมัลติทาสกิ้ง

LDT ถ้าถูกกำหนดให้ใช้งาน สามารถถูกใช้ร่วมกับ GDT เพื่อขยายจำนวนของเดสคริปเตอร์และช่วงที่สามารถอ้างแอดเดรสได้ของงานที่ถูกเลือก โดยปกติแล้วงานแต่ละงานจะมี LDT ของตัวเอง แต่งานหลาย ๆ งานก็สามารถใช้ LDT หลาย ๆ ตารางร่วมกันได้ LDT หลาย ๆ ตารางไม่จำเป็นต้องถูกกำหนดทั้งหมด ถ้าระบบไม่ต้องการสิ่งนั้นหรือฮาร์ดแวร์ที่สนับสนุนระบบมัลติทาสกิ้งไม่ถูกใช้งาน สำหรับ LDT ที่ถูกใช้ในระบบมัลติทาสกิ้งจะได้กล่าวในเรื่องของมัลติทาสกิ้งต่อไป รูปที่ 6 จะแสดงความสัมพันธ์ระหว่างเดสคริปเตอร์ ตารางของเดสคริปเตอร์ และบริเวณของแอดเดรสเชิงเส้นที่มันบ่งถึง

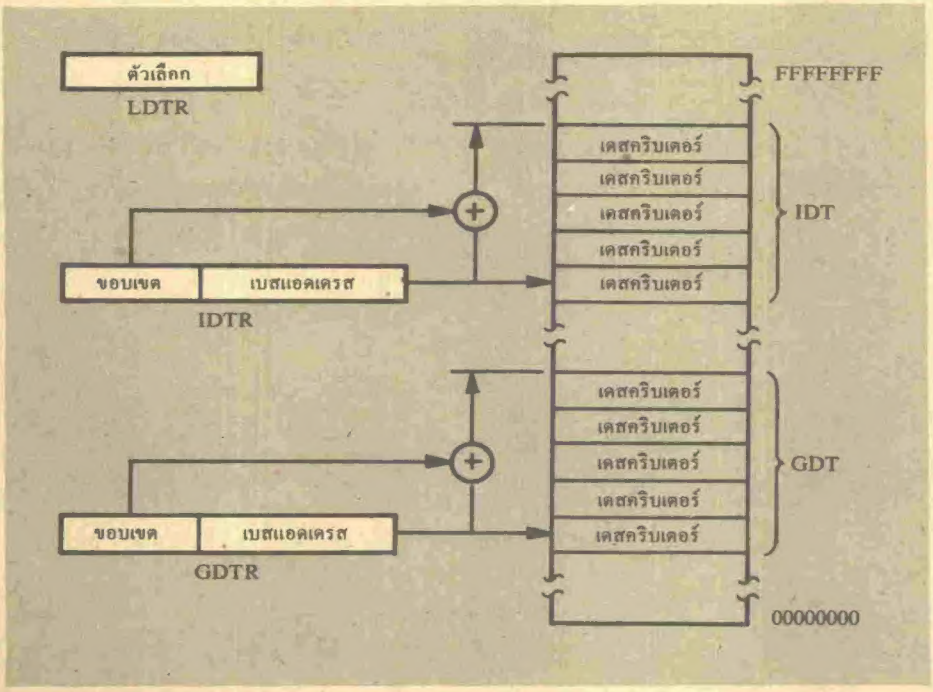
หลังจากที่เรานำแต่ละเดสคริปเตอร์ไว้ในตารางเดสคริปเตอร์แล้ว จะต้องบอกให้ตัวโปรเซสเซอร์รู้ว่าตารางเดสคริปเตอร์นี้อยู่ที่ไหน ซึ่งเราสามารถที่จะนำตารางนี้ไปไว้ตำแหน่งใดก็ได้ในพื้นที่ของหน่วยความจำ และไม่จำเป็นต้องไว้รวมกัน เพื่อที่จะให้โปรเซสเซอร์วางตำแหน่งของ GDT IDT และ LDT (ถ้ามี) เราจะต้องโหลดรีจิสเตอร์พิเศษของ 80386 คือ GDTR IDTR และ LDTR ตามลำดับ แต่ละรีจิสเตอร์มีขนาดใหญ่พอที่จะเก็บเบสแอดเดรสขนาด 32 บิตของตารางเดสคริปเตอร์และค่าขอบเขตของตาราง เบสแอดเดรสของตารางเดสคริปเตอร์ก็คือ แอดเดรสที่เป็นเชิงเส้นของไบต์แรกของเดสคริปเตอร์ตัวแรกในตาราง ขอบเขตของตารางจะต้องถูกกำหนดลงไปเพื่อโปรเซสเซอร์จะสามารถบอกความยาวของตารางและจำนวนของเดสคริปเตอร์ที่มีอยู่ได้ เนื่องจากเดสคริปเตอร์ยาว 8 ไบต์ ดังนั้นขอบเขตของตารางจะมีค่าเท่ากับ (เดสคริปเตอร์ (ไบต์/8-1) เช่นมี 1 เดสคริปเตอร์ ก็จะมีค่าขอบเขตเท่ากับ (8/8)-1=0

ตัวเลือกเซกเมนต์

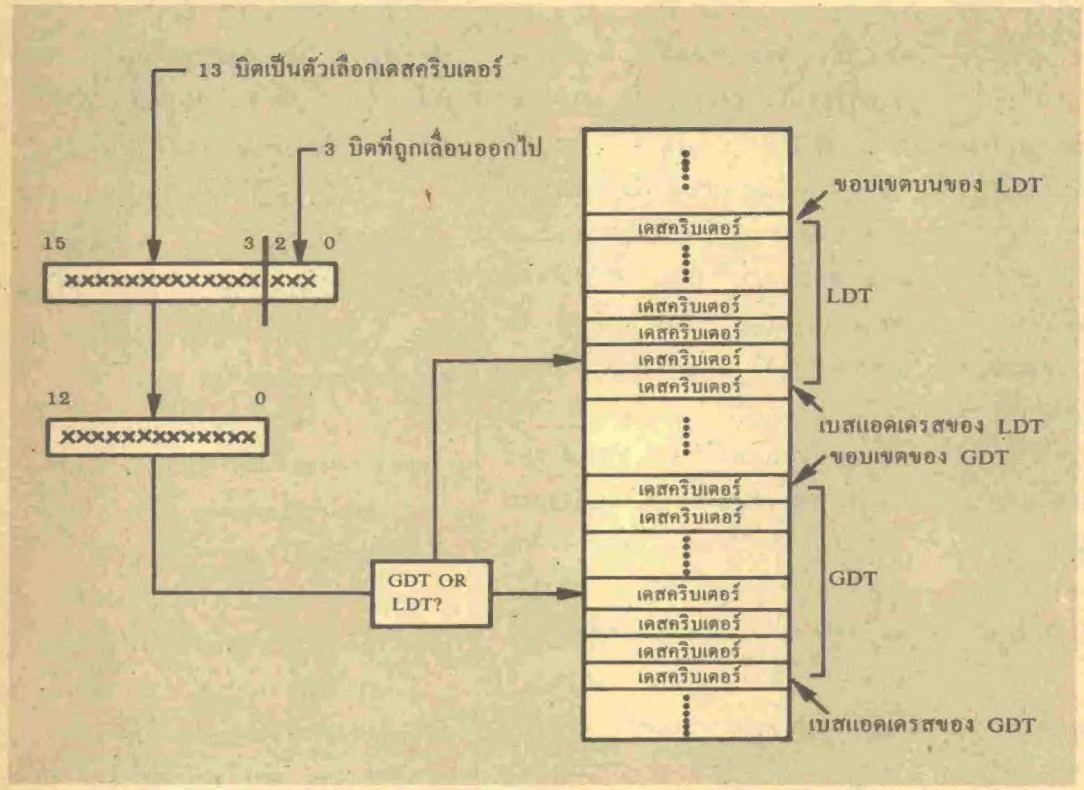
เมื่อเดสคริปเตอร์ถูกกำหนดขึ้น ตัวโปรเซสเซอร์จะใช้มันอย่างไร ผู้ใช้ที่ต้องการเขียนโปรแกรมเพื่อรัน จะกระทำกับมันอย่างไร สิ่งนี้เป็นส่วนหนึ่งของการจัดการกับหน่วยความจำของ 80386 ที่จะต้องทำให้คอมแพทติเบิ้ลกับ 8086 ทั้ง ๆ ที่กลไกของการแบ่งเซกเมนต์ไม่คอมแพทติเบิ้ลกันอย่างสมบูรณ์

ถึงแม้ว่ากลไกในการแบ่งเซกเมนต์จะเปลี่ยนไปค่อนข้างมาก และเซกเมนต์รีจิสเตอร์ก็ยังคงเป็นขนาด 16 บิตเหมือนเดิม และที่น่าแปลกอีกอย่างหนึ่งก็คือการเลือกเซกเมนต์ของ 80386 มีส่วนที่คล้ายกับ 8086 เราลองนึกย้อนไปถึง 8086 คงพอจะจำได้ว่าเบสแอดเดรสของเซกเมนต์ได้มาจากการเลื่อนค่าในเซกเมนต์รีจิสเตอร์ไปทางซ้าย 4 บิต (เป็นการคูณด้วย 16) แต่สำหรับ 80386 เซกเมนต์รีจิสเตอร์จะถูกเลื่อนบิตไปทางขวา 3 บิต (เป็นการหารด้วย 8) เพื่อเลือกเดสคริปเตอร์ตัวใดตัวหนึ่งในตารางของเดสตริปเตอร์ เช่นถ้าเราใส่ค่าใดค่าหนึ่งเข้าไปในเซกเมนต์รีจิสเตอร์ แล้วเลื่อนไปทางขวา 3 บิต เราจะได้ค่าค่าหนึ่งสมมติให้เป็น N ซึ่งได้มาจาก 13 บิต ซึ่งค่า N นี้จะเป็นค่าที่ชี้ไปที่เดสคริปเตอร์ในตารางของเดสคริปเตอร์

เราจะเห็นว่าความสัมพันธ์ระหว่างเซกเมนต์รีจิสเตอร์และเซกเมนต์เดสคริปเตอร์ใน 80386 ก็คือเซกเมนต์รีจิสเตอร์ทำตัวเป็นพอยน์เตอร์ขนาด 13 บิต ชี้ไปที่อะเรย์ของเซกเมนต์เดสคริปเตอร์ เนื่องจากพอยน์เตอร์มีขนาด 13 บิต ค่าของมันจึงอยู่ระหว่าง 0 ถึง 8191 ซึ่งก็เป็นคำตอบว่าทำไมตารางเดสคริปเตอร์ได้สูงสุด 8192 โดยแต่ละเดสคริปเตอร์มีขนาด 8 ไบต์ จึงทำให้ตารางเดสคริปเตอร์มีขนาด 64 กิโลไบต์ แต่จะมีคำถามอีกว่าทำไมจึงใช้เพียง 13 บิตแทนที่จะใช้ 16 บิต ซึ่ง 3 บิตของเซกเมนต์รีจิสเตอร์ที่เลื่อนออกไป จะไม่ทำให้ค่าในเซกเมนต์ 8 ค่าที่ต่อกันชี้ไปยังเซกเมนต์เดสคริปเตอร์ตัวเดียวกันหรือ และตารางเดสคริปเตอร์ใดที่เซกเมนต์รีจิสเตอร์อ้างถึง ดังแสดงในรูปที่ 7



รูปที่ 6 แสดงเดสคริปเตอร์และตารางของเดสคริปเตอร์



รูปที่ 7
แสดงการเลือกเดสคริปเตอร์และตารางเดสคริปเตอร์

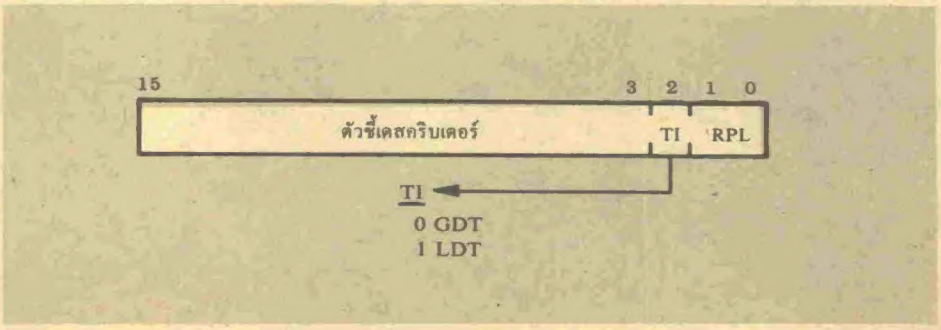
จากรูปที่ 7 จะเห็นว่าไม่ว่า 3 บิตที่ถูกเลื่อนออกไป จะเป็น 000 001... 111 ก็ตาม 13 บิตที่เหลือก็มีค่าเท่าเดิม ซึ่งจะทำให้ชี้ไปที่เดสคริปเตอร์ตัวเดียวกัน เราสามารถจะหาคำตอบได้โดยมาดูว่า 3 บิตล่างของเซกเมนต์รีจิสเตอร์ใช้สำหรับอะไร โดยความเป็นจริงแล้ว 3 บิตล่างมีความสำคัญสำหรับการแบ่งเซกเมนต์และการป้องกันทางฮาร์ดแวร์ของ 80386 บิตที่ 0 และบิตที่ 1 ใช้สำหรับกำหนดระดับสิทธิพิเศษมีค่าจาก 0 ถึง 3 ดังนั้น 2 บิตนี้จึงไม่มีผลต่อเดสคริปเตอร์ที่เซกเมนต์รีจิสเตอร์ชี้ไป ส่วนบิตที่ 2 เป็นตัวบอกว่าจะอ้างถึงตารางเดสคริปเตอร์ใด ถ้าบิตนี้เป็น 0 เซกเมนต์รีจิสเตอร์จะอ้างถึงเดสคริปเตอร์ที่ N ใน GDT แต่ถ้าเป็น 1 จะอ้างถึงเดสคริปเตอร์ที่ N ใน LDT ดังนั้นคำตอบข้างต้นก็คือ ค่าในเซกเมนต์ 8 ค่าที่อยู่ติดกันจะไม่ชี้เดสคริปเตอร์ตัวเดียวกัน แต่ 4 ค่าที่อยู่ติดกันจะชี้เดสคริปเตอร์ตัวเดียวกัน

ในกรณีของ 8086 ค่าใด ๆ ที่ถูกโหลดไว้ในเซกเมนต์รีจิสเตอร์ จะกลายเป็นเบสแอดเดรส แต่สำหรับ 80386 ค่าใด ๆ ในเซกเมนต์รีจิสเตอร์จะเป็นตัวชี้ไปยัง GDT หรือLDT เราจะเรียกค่า 16 บิตในเซกเมนต์รีจิสเตอร์ว่าตัวเลือกเซกเมนต์ เพราะว่ามันเลือกเซกเมนต์เดสคริปเตอร์จากตารางของเดสคริปเตอร์ ตำแหน่งในตัวเลือกมีดังนี้ และแสดงดังรูปที่ 8

INDEX (Descriptor index, บิตที่ 3 ถึงบิตที่ 15) : ค่านี้จะเลือกค่าใดค่าหนึ่งในตารางเดสคริปเตอร์ซึ่งเป็นไปได้ทั้งหมด 8192 ค่า (เดสคริปเตอร์)

TI (Table indicator, บิตที่ 2) : เมื่อบิตนี้เป็น 0 ค่าในอินเด็กซ์จะอ้างถึงเดสคริปเตอร์ใน GDT ถ้าเป็น 1 จะอ้างถึงเดสคริปเตอร์ใน LDT

RPL (Request Privilege Level, บิตที่ 0 และบิตที่ 1) : 2 บิตนี้ใช้สำหรับการตรวจสอบระดับสิทธิพิเศษซึ่งเป็นกลไกของการป้องกัน



รูปที่ 8 รูปแบบของตัวเลือกเซกเมนต์

รูปที่ 9 แสดงเส้นทางการเปลี่ยนจากลอจิกคัลแอดเดรสไปเป็นแอดเดรสเชิงเส้น เริ่มต้นเราจะโหลดค่าไว้ใน DS จะเห็นว่าบิตที่ TI เป็น 0 ดังนั้นโปรเซสเซอร์จะใช้ส่วนของอินเด็กซ์ (บิตที่ 3-15) ชี้ไปยัง GDT เพื่อหาเดสคริปเตอร์ของดาต้าเซกเมนต์ ค่าเบสแอดเดรสรวมทั้งค่าขอบเขตของ GDT สามารถถูกบอกตำแหน่งโดยผ่าน GDTR ตัวโปรเซสเซอร์จะนำเบสแอดเดรสที่แยกกันอยู่ในเดสคริปเตอร์ (บิตที่ 16-39 และบิตที่ 56-63) มาต่อกันเป็นเบสแอดเดรสของเซกเมนต์ DS ซึ่งจะถูกรวมกับค่าออฟเซตในรีจิสเตอร์ ESI เป็นค่าแอดเดรสเชิงเส้นที่เป็นตำแหน่งจริงในหน่วยความจำ

ในขณะที่เวลาเป็นสิ่งที่ต้องให้พิจารณาและเพื่อลดการติดต่อกับหน่วยความจำภายนอก 80386 ไม่ได้ทำงานดังกล่าวข้างต้นทุกครั้งที่อ้างถึงหน่วยความจำ แต่มันจะทำการเก็บเดสคริปเตอร์ทั้ง 8 ไบต์ไว้ในหน่วยความจำแคชเมื่อมันโหลดตัวเลือกเดสคริปเตอร์ไว้ในเซกเมนต์รีจิสเตอร์ตั้งแต่ครั้งแรก ทำให้มันมีข้อมูลที่สามารถอ้างได้รวดเร็วจนกว่าค่าในเซกเมนต์รีจิสเตอร์จะถูกเปลี่ยนไป

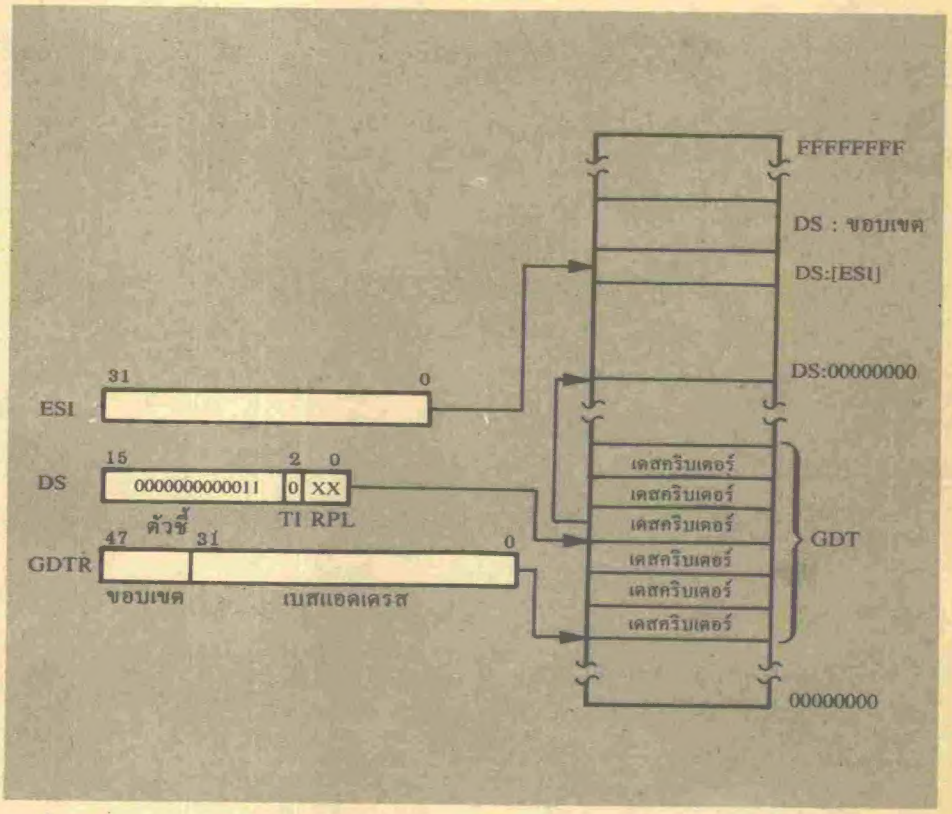
การโหลดตัวเลือกเซกเมนต์

อย่างที่กล่าวไปข้างต้น ค่าที่โหลดไว้ในเซกเมนต์รีจิสเตอร์ (ตัวเลือกเดสคริปเตอร์) มีความสัมพันธ์กับเดสคริปเตอร์โดยตรง ตัวเลือกใด ๆ จะเลือกเพียงหนึ่งเดสคริปเตอร์เท่านั้น อย่างไรก็ตามค่าของตัวเลือกที่ต่อกัน 4 ค่าจะเลือกเดสคริปเตอร์ตัวเดียวกัน เพราะว่าฟิลด์ RPL (บิตที่ 0 และ 1) ไม่มีผลต่อการเลือกเดสคริปเตอร์ การเปลี่ยนเพียงไม่กี่บิตในเซกเมนต์รีจิสเตอร์จะมีความสำคัญมากต่อความสามารถในการอ้างหน่วยความจำ จะเห็นได้ว่าค่าต่าง ๆ มากมายสามารถใช้ได้กับ 8086 แต่กลับไม่มีความหมายเลยกับ 80386 ในโปรเทคโหมด

เมื่อใดก็ตามที่เซกเมนต์รีจิสเตอร์ถูกเปลี่ยน ไม่ว่าทางตรงโดยโปรแกรมของเราหรือทางอ้อม เช่น การเกิดอินเตอร์รัพต์ 80386 จะทำการตรวจสอบตัวเลือกก่อนที่จะโหลดค่าลงไปในเซกเมนต์รีจิสเตอร์ เช่น การตรวจสอบระดับสิทธิพิเศษ และอื่น ๆ เพื่อป้องกันการโหลดตัวเลือกที่ไม่ถูกต้อง

ในทุกกรณี ตัวโปรเซสเซอร์จะตรวจสอบดูว่าฟิลด์อินเด็กซ์ของตัวเลือกว่าอยู่ในขอบเขตของตารางเดสคริปเตอร์ที่แสดงโดยบิต TI หรือไม่ นั่นคือเหตุผลที่ว่าทำไมค่าขอบเขตของตารางเดสคริปเตอร์จึงอยู่กับค่าเบสแอดเดรสของมัน ถ้าตัวเลือกผ่านการตรวจสอบเหล่านี้ 80386 จะตรวจสอบแฟล็ก Present (บิตที่ 47) ของเดสคริปเตอร์ เพื่อดูว่าเซกเมนต์นั้นอยู่ที่นั่นจริง ๆ ถ้าใช่มันจึงโหลดค่าตัวเลือกนั้นลงไป และเดสคริปเตอร์ที่อ้างถึงจะถูกโหลดไว้ในหน่วยความจำแคช ถ้าการตรวจสอบใด ๆ นี้ไม่ผ่าน 80386 จะเกิดเอ็กเซพชันขึ้นและไม่มีสิ่งใดที่ถูกเปลี่ยนแปลง

พิจารณาความสัมพันธ์ระหว่างตัวเลือกและเดสคริปเตอร์ที่ตัวเลือกชี้ไป และบริเวณของแอดเดรสที่เดสคริปเตอร์ชี้ไป ประการแรกก็คือการเพิ่มค่าในเซกเมนต์รีจิสเตอร์ ถ้าเป็นการเพิ่มค่าในฟิลด์ RPL จะต้องไม่ทำให้ตัวเลือกชี้ไปยังเดสคริปเตอร์ตัวต่อไปในตารางเดสคริปเตอร์ ซึ่งการเพิ่มนี้จะมีผลต่อกลไกของการตรวจสอบระดับสิทธิพิเศษเท่านั้น ถ้าเราต้องการเลือกเดสคริปเตอร์ตัวต่อไป จะต้องเพิ่มค่าตัวเลือกทีละ 8 ประการที่สองก็คือ เดสคริปเตอร์ที่อยู่ติดกันในตาราง ไม่มีความสัมพันธ์กัน



รูปที่ 9 การอ้างแอดเดรสในหน่วยความจำ

เลย (ถ้าเราไม่ได้กำหนดขึ้นมา) ดังนั้นการเพิ่มค่าตัวเลือกเพื่อชี้เดสคริปเตอร์ตัวต่อไป ก็อาจจะไม่ใช่ช่วงของแอดเดรสที่ต่อจากตัวก่อนหน้า โดยสรุปก็คือไม่มีสิ่งที่เด่นชัดใด ๆ ที่จะบอกได้ว่าค่าของตัวเลือกจะชี้ไปยังแอดเดรสบริเวณใด

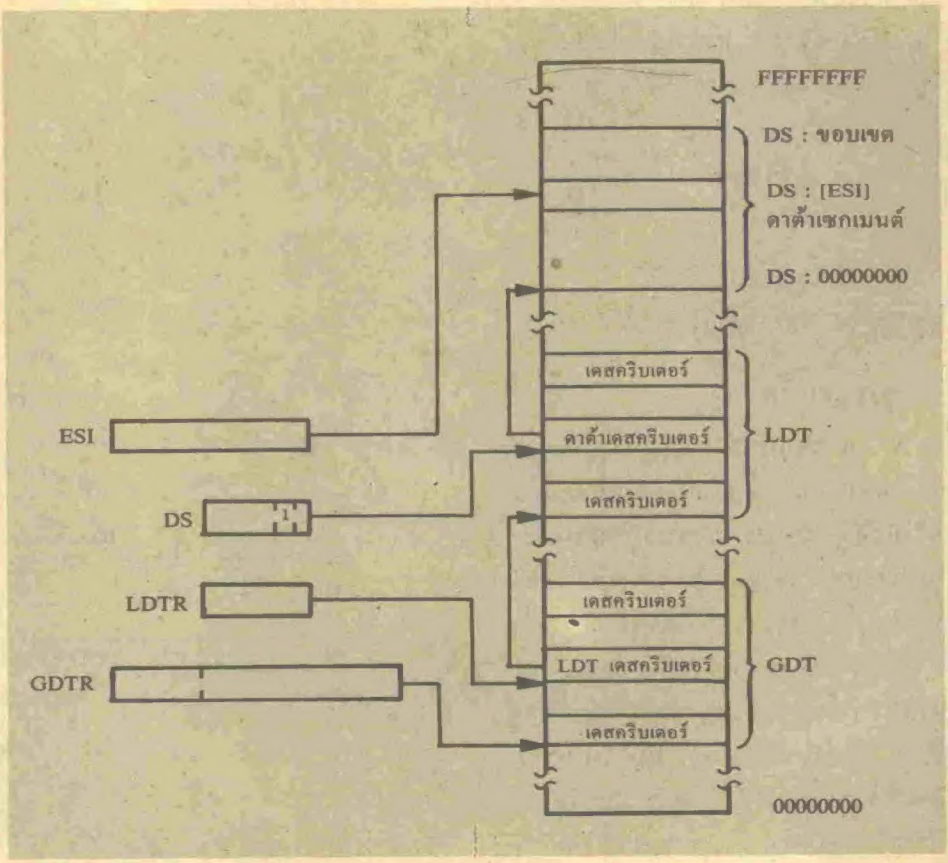
จากข้อมูลทั้งหมดนี้เราจะรู้สึกว่างานของนักเขียนโปรแกรมระดับแอสเซมบลีเกือบจะหมดหวังไปเลย เราไม่สามารถจัดการกับเซกเมนต์รีจิสเตอร์อย่างแน่นอนถ้าเราไม่มีความรู้ในการจัด GDT และ LDT เป็นอย่างดี

ตารางโลคอลเดสคริบเตอร์ (LDT)

อย่างที่ทราบมาแล้วว่ามีตารางเดสคริบเตอร์อยู่ 3 ชนิด คือ โกบอลโลคอล และอินเทอร์รัพ ในขณะที่โปรเซสเซอร์กำลังรันอยู่ในโปรเทคโหมดจะต้องมีตาราง GDT และ IDT อย่างละหนึ่งตารางอย่างแน่นอน เราสามารถกำหนดตารางอื่นในหน่วยความจำได้ แต่มีเพียงหนึ่งตารางเท่านั้นที่ถูกพิจารณาว่ามีแอคตีฟในขณะใดขณะหนึ่ง ตารางอันหนึ่งจะถูกชี้โดยรีจิสเตอร์ GDTR และ IDTR ดังนั้นเราสามารถกำหนดตารางโลคอลเดสคริบเตอร์ตั้งแต่ 0 ถึง 8191 เดสคริบเตอร์ได้ถึงแม้ว่าจะมีตารางอื่นอยู่ และเพียงหนึ่งตารางเท่านั้นที่แอคตีฟในขณะหนึ่ง

LDTs กระทำตัวเหมือนกับส่วนขยายของ GDT แต่มันจะถูกกำหนดให้ใช้สำหรับงานใดงานหนึ่ง เมื่อมีการใช้ฮาร์ดแวร์สนับสนุนสำหรับการสลับงานโปรแกรมขณะที่กำลังรันอยู่นั้น สามารถใช้เดสคริบเตอร์ใน GDT และใน LDT ที่ถูกกำหนดให้กับงานนั้นได้ ถ้า LDT ไม่ถูกกำหนดให้ใช้เดสคริบเตอร์ทุกตัวจะมาจาก GDT และเราจะต้องหลีกเลี่ยงการใช้ตัวเลือกที่มีบิต TI = 1 (บิตที่ 2) ดังนั้นการที่มีการกำหนด LDT สำหรับงานหนึ่งจะทำให้เกิดการขยายช่วงของแอดเดรสอย่างมีประสิทธิภาพ คุณลักษณะอันนี้สามารถถูกใช้เพื่อยอมให้เพียงงานหนึ่งที่แน่นอนแอคเซสกับบริเวณของหน่วยความจำพิเศษ เช่น บัฟเฟอร์ของจอภาพ หรือดิสค์คอนโทรลเลอร์

LDTs ถูกสร้างขึ้นได้เช่นเดียวกับ GDTs หรือ IDTs อย่างไรก็ตามวิธีการวางตำแหน่งของมันจะแตกต่างกันอย่างมาก ขณะที่ 80386 มีรีจิสเตอร์พิเศษขนาด 48 บิต เพื่อเก็บค่าเบสแอดเดรสและขอบเขตของ GDT และ IDT (โดยปกติแล้วจะไม่เปลี่ยนค่าบ่อยครั้ง) LDT ตัวปัจจุบัน (ค่าอาจจะเปลี่ยนทุกครั้งที่มีการเปลี่ยนงาน) จะถูกชี้โดยตัวเลือก 16 บิต ตัวเลือกนี้ (LDTR) ต้องอ้างถึงเดสคริบเตอร์พิเศษของ LDT ใน GDT โดยวิธีเดียวกันที่เดสคริบเตอร์ของโค้ดเซกเมนต์กำหนดขอบเขตบนและล่างของเซกเมนต์ที่เก็บโค้ด เดสคริบเตอร์ของ LDT ใน GDT ก็กำหนดขอบเขตบนและล่างของเซกเมนต์ที่เก็บโค้ดเดสคริบเตอร์ของ LDT ใน GDT ก็กำหนดเบสแอดเดรสและขอบเขตของตารางเดสคริบเตอร์ของ LDT GDT สามารถเก็บเดสคริบเตอร์ของ LDT จำนวนเท่าไรก็ได้ หนึ่งเดสคริบเตอร์สำหรับหนึ่ง LDT ที่จะถูกกำหนด โดยวิธีนี้โปรเซสเซอร์สามารถติดตาม LDT ตัวปัจจุบันได้โดยการโหลดตัวเลือกซึ่งชี้ไปที่เดสคริบเตอร์ที่กำหนดตำแหน่งและขนาดของตารางเดสคริบเตอร์ของอีกตัวหนึ่ง การอ้างหน่วยความจำใด ๆ ที่ใช้เซกเมนต์ที่มีตัวเลือกโดยมีบิต TI = 1 จะถูกผ่าน LDT รูปที่ 10 แสดงตัวอย่างการอ้างทางอ้อม 2 ครั้งของ LDT



รูปที่ 10 ตารางโลคอลเดสคริบเตอร์

เดสคริปเตอร์ของ LDT เป็นตัวอย่างอันหนึ่งของเซกเมนต์เดสคริปเตอร์ของระบบ เมื่อแฟล็ก S (บิตที่ 44) ในเดสคริปเตอร์ไม่ถูกเซต มันจะแสดงว่าเป็นเซกเมนต์เดสคริปเตอร์ของระบบ ซึ่งตรงข้ามกับเดสคริปเตอร์ของโค้ดดาต้าหรือของสแต็ก เซกเมนต์เดสคริปเตอร์ของระบบจะไม่มีบิต Accessed แต่ฟิลด์ของชนิด (type) จะเพิ่มเป็น 4 บิต ถ้าค่าในฟิลด์ Type เป็น 2 เดสคริปเตอร์เป็นเดสคริปเตอร์ของ LDT ซึ่งแสดงดังรูปที่ 11 ฟิลด์ของเบสแอดเดรสเป็นแอดเดรสเชิงเส้นขนาด 32 บิต สำหรับ LDT และฟิลด์ของขอบเขตกำหนดความยาว LDT ลบด้วย 1 เช่นเดียวกับใน GDT และ IDT สังเกตอีกว่าค่าขอบเขตมีได้ 20 บิตรวมกับบิต G อีกหนึ่งบิต ทำให้เรากำหนดขนาดของ LDT ได้มากกว่า 64 กิโลไบต์

เพื่อที่จะใช้บริการของ LDT เราก็ทำได้ง่ายโดยโหลด LDTR ด้วยตัวเลือกที่เหมาะสมสำหรับเดสคริปเตอร์ของ LDT เพื่อใช้สำหรับ LDT โดยลำดับแล้ว LDTR ก็เหมือนกับเซกเมนต์รีจิสเตอร์อันอื่น ๆ ถ้าเราพยายามจะโหลด LDTR ด้วยค่าใด ๆ ที่ไม่ใช่ตัวเลือกไปยังเดสคริปเตอร์ของ LDT 80386 จะเกิดโปรเทคชันฟอลท์ขึ้น

สำหรับ IDT จะไม่กล่าวถึงในที่นี้ เพราะว่าโปรแกรมจะไม่เคยอ้างถึงเดสคริปเตอร์ใน IDT เลย ซึ่ง IDT นี้จะมีเพื่อให้ 80386 หาเอ็กเซพชันที่เหมาะสมที่จัดการโดยตัวมันเอง

การเรียกชื่ออื่นและการซ้อนกันของเซกเมนต์

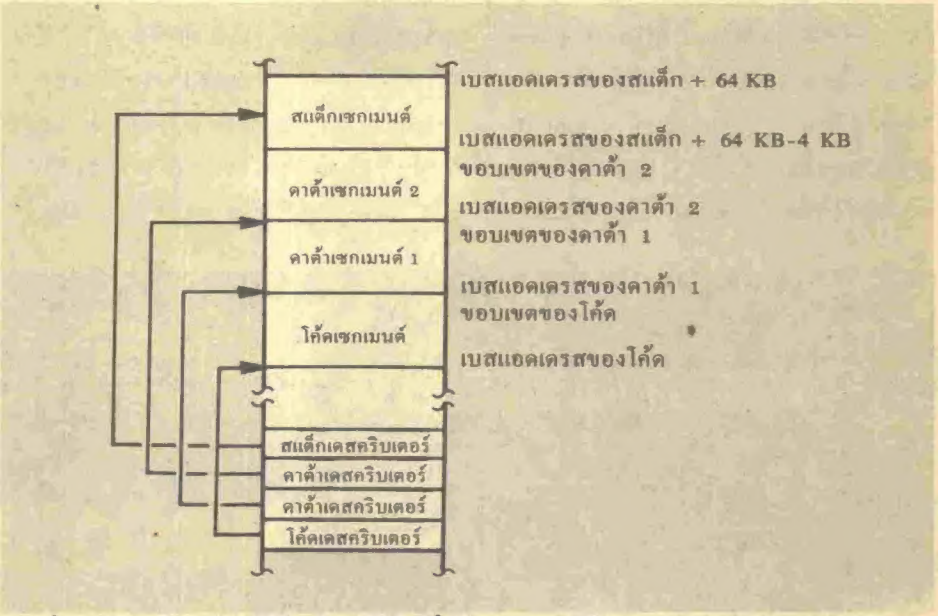
80386 อนุญาตให้เราสร้างเซกเมนต์ซึ่งเป็นชนิดอ่านได้อย่างเดียว เอ็กซีคิวท์ได้อย่างเดียว อ่านได้/เขียนได้ หรือเอ็กซีคิวท์ได้/อ่านได้ จะไม่มีการนำมารวมกันเหมือนอย่างกรณีของ 8086 ที่สามารถอ่านได้ เขียนได้ และเอ็กซีคิวต์ได้พร้อมกันในเซกเมนต์เดียว อย่างไรก็ตามเพื่อให้ได้ผลเช่นนี้ สามารถทำได้โดยวิธีอื่น

โดยปกติแล้วเซกเมนต์เดสคริปเตอร์จะถูกสร้างขึ้นขณะที่มีการอินนีเชียไลซ์ระบบ เพื่อกำหนดพื้นที่ของหน่วยความจำที่เราคาดว่าจะใช้ เช่น โค้ด ดาต้า หรือสแต็ก เราต้องกำหนดโค้ดเซกเมนต์ขึ้นมาอันหนึ่งเพื่อเก็บส่วนที่เอ็กซีคิวท์ได้ ดาต้าเซกเมนต์อีกหนึ่งหรือสองเซกเมนต์ และสแต็กเซกเมนต์ที่ใหญ่เพียงพอ เราสามารถกำหนดให้เซกเมนต์เหล่านี้เป็นแอดเดรสที่อยู่ต่อกันได้ โดยท้ายของเซกเมนต์หนึ่งอยู่ติดกับต้นของเซกเมนต์อีกอันหนึ่ง

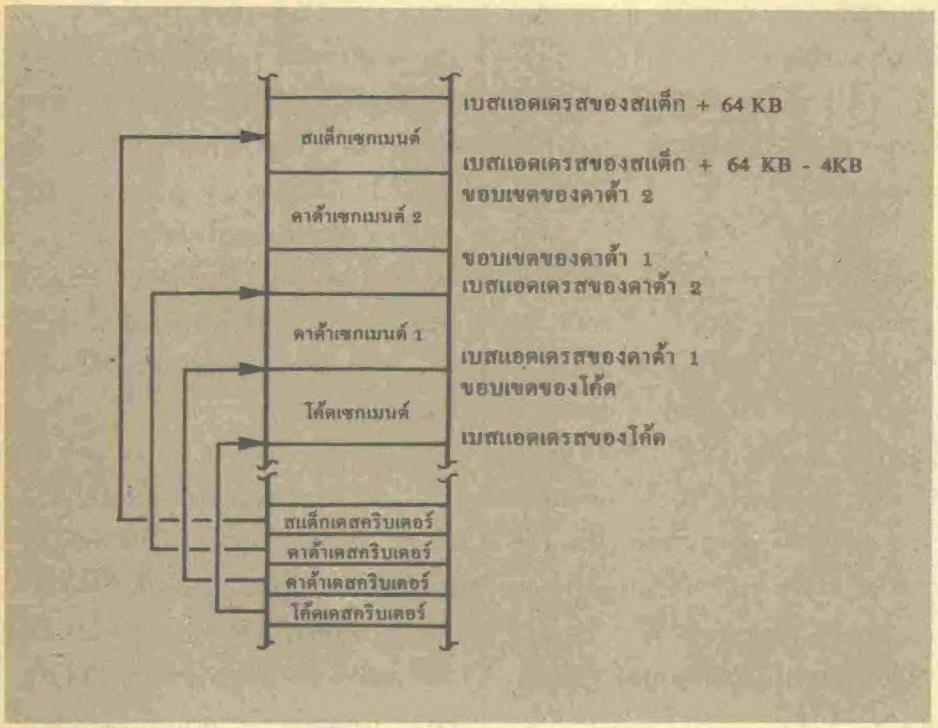
รูปที่ 12 แสดงการอยู่ต่อกันของเซกเมนต์เหล่านี้ โดยมีโค้ดเซกเมนต์ขนาด 6 กิโลไบต์อยู่ส่วนล่างของหน่วยความจำ ตามมาด้วยดาต้าเซกเมนต์ขนาด 12 กิโลไบต์ และสแต็กเซกเมนต์ขนาด 4 กิโลไบต์ต่อท้ายจากดาต้าเซกเมนต์ตัวที่สอง จะเห็นว่าเซกเมนต์เหล่านี้อยู่ต่อกันพอดีไม่ซ้อนกัน ดังนั้นจะมีค่าเซกเมนต์และออฟเซตเดียวที่อ้างถึงตำแหน่งต่าง ๆ

จะเกิดอะไรขึ้นถ้าเราเปลี่ยนเดสคริปเตอร์ของดาต้าเซกเมนต์อันแรก โดยเพิ่มขนาดของเซกเมนต์ขึ้นอีก 4 กิโลไบต์ จะเห็นว่าท้ายของดาต้าเซกเมนต์ที่หนึ่งจะเกินเข้าไปในดาต้าเซกเมนต์ที่ 2 อีก 4 กิโลไบต์ ทำให้เซกเมนต์ทั้งสอง

รูปที่ 11 รูปแบบของ LDT เดสคริบเตอร์



รูปที่ 12 เซกเมนต์ของหน่วยความจำที่ต่อกันพอดี



รูปที่ 13 เซกเมนต์ของหน่วยความจำที่ซ้อนกัน

ซ้อนกัน ดังรูปที่ 13 ซึ่ง 80386 จะไม่ถือว่าเป็นข้อผิดพลาด และยอมให้เกิดการซ้อนกันเป็นบางส่วนหรือซ้อนกันทั้งเซกเมนต์ก็ได้

เมื่อเซกเมนต์เกิดการซ้อนกัน เราสามารถจะอ้างถึงบริเวณหน่วยความจำที่ซ้อนกันได้ 2 วิธี โดยอ้างออฟเซตที่เป็น 0 ของดาต้าเซกเมนต์ที่สอง ซึ่งจะให้ผลเหมือนกับการอ้างออฟเซตที่ 3000 (12 กิโลไบต์ + 1) ของดาต้าเซกเมนต์ที่หนึ่ง ทำนองเดียวกันเราสามารถเขียนข้อมูลลงไปที่ออฟเซต ØFFF ของเซกเมนต์ที่สองและอ่านค่ากลับมาด้วยออฟเซตที่ 3FFF ของเซกเมนต์ที่หนึ่ง

ถึงแม้ว่า 80386 จะไม่ยอมให้เขียนลงไปในเซกเมนต์ที่กำหนดให้เป็นชนิดเอ็กซีคิวท์ได้อย่างเดียว แต่ก็ไม่ผิดอะไรถ้าเราจะเขียนลงไปในเซกเมนต์ที่เป็นชนิดเขียนได้ ซึ่งครอบคลุมแอดเดรสบริเวณเดียวกับที่เอ็กซีคิวท์ได้ ถ้าเรากำหนดเดสคริปเตอร์ของโค้ดเซกเมนต์และดาต้าเซกเมนต์ให้มีเบสแอดเดรสและขอบเขตเดียวกัน เราสามารถที่จะเขียนข้อมูลลงไปในบริเวณของโค้ดและสามารถเอ็กซีคิวท์ในบริเวณของดาต้า ซึ่งวิธีนี้จะเรียกว่าการเรียกชื่ออื่นของเซกเมนต์

การซ้อนกันและการเรียกชื่ออื่นของเซกเมนต์จะเป็นประโยชน์อย่างมาก เมื่อระบบจัดการ (OS) ต้องการให้โปรแกรมใช้ส่วนของโครงสร้างข้อมูลร่วมกัน การเรียกชื่อของดาต้าเซกเมนต์แทนโค้ดเซกเมนต์เป็นตัวอย่างอันหนึ่งที่ดี โดยจะยอมให้เราเขียนโค้ดที่สามารถเปลี่ยนแปลงตัวเองได้

สรุป

การจัดหน่วยความจำของ 80386 ซึ่งเป็นแบบเซกเมนต์เช่นเดียวกับ 8086 นั้น แต่ความจริงแล้วกลไกลในการอ้างแอดเดรสจะแตกต่างจาก 8086 ค่อนข้างมาก เซกเมนต์ของ 80386 จะมีขนาดเท่าไรก็ได้ (มากที่สุดคือ 4 กิกะไบต์) และไม่จำเป็นต้องมีขนาดเท่ากัน แต่ละเซกเมนต์เดสคริปเตอร์เป็นตัวบอกลักษณะของมัน เช่น เบสแอดเดรส ขอบเขต ชนิด ระดับสิทธิพิเศษ และอื่น ๆ เดสคริปเตอร์ต่าง ๆ จะถูกเรียงต่อกันในตารางเดสคริปเตอร์ซึ่งมี 3 ชนิด คือ GDI IDT และ LDT แต่ละชนิดก็จะถูกใช้งานต่าง ๆ กันไปเช่น IDT จะถูกใช้เช่นเดียวกันกับตารางของอินเตอร์รัพต์เวคเตอร์ใน 8086 ส่วน LDT จะถูกใช้ต่อเมื่อมีการใช้งานแบบมัลติทาสกิ้งเท่านั้น

หนังสือที่เขียน ไมโครคอมพิวเตอร์ ไม่ควรพลาด
ยูทิลิตี้ แอดวานซ์ดีเบส